МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья.

Студентгр. 8304 _____ Мельникова О.А.

Преподаватель _____ ФирсовМ.А.

Санкт-Петербург

2019

Цель работы.

Изучить бинарные деревья и леса, реализовать бинарное дерево на векторе.

Текст задания 4-Д.

Заданы два бинарных дерева b1 и b2 типа BT с произвольным типом элементов. Проверить:
- подобны ли они (два бинарных дерева подобны, если они оба пусты либо они оба непусты и их левые поддеревья подобны и правые поддеревья подобны);
- равны ли они (два бинарных дерева равны, если они подобны и их соответствующие элементы равны);
- зеркально подобны ли они (два бинарных дерева зеркально подобны, если они оба пусты либо они оба непусты и для каждого из них левое поддерево одного подобно правому поддереву другого);
- симметричны ли они (два бинарных дерева симметричны, если они зеркально подобны и их соответствующие элементы равны).

Описание алгоритма.

Программа считывает две строки, отделяет их от внешних скобок, после чего формируются два бинарных дерева, с помощью функции makeFromString. Ее работа заключается в том, что если строка верная, в зависимости от скобок определяется поддерево элемента строки и уже для него вызывается makeFromString.

Выявление подобия происходит с помощью функции isResemlanceWith. Используется булевская переменная result, изначально равная false. При условии, что деревья не пусты, проверяются поддеревья: если всех нет, это означает, что раз алгоритм дошел до этого этапа — все хорошо. Переменной result присваивается true. Иначе проверяются следующие поддеревья с помощью рекурсии. Используется &= чтобы не потерять предыдущие результаты. Зеркальное подобие работает также, но вместо проверки подобия левого поддерева одного дерева левому поддереву другого, запускается проверка подобия левого поддерева одного дерева правому поддереву другого. Проверка на равенство работает также как и выявление подобия, но добавляется условие равенства значений элементов, это же условие

добавляется для проверки на симметричность, в остальном же эта проверка не отличается от проверки деревьев на зеркальное подобие.

Описание класса.

1. `T value;`

Хранит значение.

2. `std::unique_ptr<BinaryTree<T>> leftLeaf;`

Указатель на левое поддерево.

3. `std::unique_ptr<BinaryTree<T>> rightLeaf;`

Указатель на правое поддерево.

4. `makeFromString(std::string str)`

Функция предназначена для создания дерева из скобочной записи.

5. `bool isResemlanceWith(std::unique_ptr<BinaryTree<T>>& secondTree)`

Проверка на подобие.

6. `bool isMirrorResemlanceWith(std::unique_ptr<BinaryTree>& secondTree)`

Проверка на зеркальное подобие.

7. `bool isEqualWith(std::unique_ptr<BinaryTree>& secondTree)`

Проверка на равенство.

8. `bool isSymmetryWith(std::unique_ptr<BinaryTree>& secondTree)`

Проверка на симметрию.

9. `int getEndPos(std::string str, int startPos)`

Функция для нахождения индекса закрывающей скобки для некой открывающей.

Тестирование.

Входные данные	Выходные данные
(a(bb)) (d(ef))	For file: forest.txt Бинарные деревья: подобны, не зеркально подобны, не равны, не симметричны.
(e(r(t(rr)e)) (a(b	For file: forest.txt Неправильная запись дерева

(a(bc(d))) (a(d(#d)g))	For file: forest.txt Бинарные деревья: не подобны, зеркально подобны, не равны, не симметричны.
(a(b(l)c(#d))) (a(b(t)f(#a)))	For file: forest.txt Бинарные деревья: подобны, зеркально подобны, не равны, не симметричны.
(a(bc(d))) (a(c(#d)b))	For file: forest.txt Бинарные деревья: не подобны, зеркально подобны, не равны, симметричны.
(a(b(c)b(#c))) (a(b(c)b(#c)))	For file: forest.txt Бинарные деревья: подобны, зеркально подобны, равны, симметричны.

Вывод.

В результате работы был получен опыт по реализации бинарных деревьев, и создана программа на языке си++, удовлетворяющая требованиям.

Приложение А.

Файл main.cpp

```
#include "BinaryTree.h"

std::string getStringWithoutBrackets(std::string str)
{
   int i = 0;
   while (str[i] == ' ') i++;
   if (str[i] != '('){
   std::cout<<"Неправильная запись дерева" << std::endl;
            exit(1);
   }
   i++;
   int br = 1;
   int symbolsCount = 0;
   while ((br > 0) && (i+symbolsCount < str.length()))
   {
      symbolsCount++;
      if (str[i+symbolsCount] == '(') br++;
      if (str[i+symbolsCount] == ')') br--;
   }
   if (br > 0){
            std::cout<<"Неправильная запись дерева" << std::endl;
      exit(1);
   }
   return str.substr(i, symbolsCount);
}


int main(int argc, char* argv[]) {
   int testCounter = 0;
   std::string str1,
               str2;
   if(argc == 1)
   {
      std::cout << "Input first string:" << std::endl;
      std::getline(std::cin, str1);
      std::cout << "Input second string:" << std::endl;
      std::getline(std::cin, str2);
   }
   else
   {
      std::cout << "For file: " << argv[1] << std::endl;
      std::ifstream inputFile(argv[1]);
      if (!inputFile.is_open())
      {
         std::cout << "ERROR: file isn't open" << std::endl;
         return 0;
      }
      if (inputFile.eof())
      {
         std::cout << "ERROR: file is empty" << std::endl;
```

```
            return 0;
        }
        std::getline(inputFile, str1);
        std::getline(inputFile, str2);
    }

    std::unique_ptr<BinaryTree<char>> tree1(new BinaryTree<char>());
    std::unique_ptr<BinaryTree<char>> tree2(new BinaryTree<char>());
    tree1->makeFromString(getStringWithoutBrackets(str1));
    tree2->makeFromString(getStringWithoutBrackets(str2));
    std::cout << "Бинарные деревья: ";
    if(tree1->isResemlanceWith(tree2)){
        std::cout << "подобны, ";
    }else{
        std::cout << "не подобны, ";
    }
    if(tree1->isMirrorResemlanceWith(tree2)){
        std::cout << "зеркально подобны, ";
    }else{
        std::cout << "не зеркально подобны, ";
    }
    if(tree1->isEqualWith(tree2)){
        std::cout << "равны, ";
    }else{
        std::cout << "не равны, ";
    }
    if(tree1->isSymmetryWith(tree2)){
        std::cout << "симметричны." << std::endl;
    }else{
        std::cout << "не симметричны." << std::endl;
    }

    return 0;
}
```

Приложение Б.

Файл BinaryTree.h

```
#pragma once
#include <iostream>
#include <string>
#include <memory>
#include <cstdlib>
#include <cstdio>
#include <fstream>

template <typename T>
class BinaryTree
{
public:
     T value;
   std::unique_ptr<BinaryTree<T>> leftLeaf;
   std::unique_ptr<BinaryTree<T>> rightLeaf;

   int makeFromString(std::string str)
   {
      if ((str.length() == 0) || (str[0] == '#'))
      {
         value = NULL;
         return 0;
      }
      if ((str[0] == '(') || (str[0] == ')')){
               std::cout<<"Неправильная запись дерева" << std::endl;
               exit(1);
      }

      value = str[0];

      std::string leftStr = "", rightStr = "";

      if (str.length() > 1)
      {
         if (str[1] != '('){
                        std::cout<<"Неправильная запись дерева" << std::endl;
                        exit(1);
                  }
         int leftStart = 2;
         int leftEnd;
         int rightStart;
         int rightEnd;

         leftEnd = getEndPos(str, leftStart);
         leftStr = str.substr(leftStart, leftEnd - leftStart + 1);

         rightStart = leftEnd + 1;
         //if (str.Length() > rightStart)
         if (str[rightStart] == ')')
            rightEnd = rightStart - 1;
         else
            rightEnd = getEndPos(str, rightStart);
         rightStr = str.substr(rightStart, rightEnd - rightStart + 1);
```

```
    }
    // a(b(c)e(#h))
    // 012345678901

    leftLeaf = std::make_unique<BinaryTree<char>>();
    rightLeaf = std::make_unique<BinaryTree<char>>();

    leftLeaf->makeFromString(leftStr);
    rightLeaf->makeFromString(rightStr);
}

bool isResemlanceWith(std::unique_ptr<BinaryTree<T>>& secondTree)
{
    bool result = false;
    if (secondTree != NULL)
    {
        if ( (secondTree->leftLeaf  == NULL) == (leftLeaf  == NULL) &&
             (secondTree->rightLeaf == NULL) == (rightLeaf == NULL)    ){
            result = true;
            if (leftLeaf != NULL)
                result = leftLeaf->isResemlanceWith(secondTree->leftLeaf);
            if (rightLeaf != NULL)
                result &= rightLeaf->isResemlanceWith(secondTree->rightLeaf);
        }
                }
                return result;
}

bool isMirrorResemlanceWith(std::unique_ptr<BinaryTree>& secondTree)
{
    bool result = false;
    if (secondTree != NULL)
    {
        if ( (secondTree->leftLeaf  == NULL) == (rightLeaf  == NULL) &&
             (secondTree->rightLeaf == NULL) == (leftLeaf   == NULL)    ){
            result = true;
            if (leftLeaf != NULL)
                result = leftLeaf->isMirrorResemlanceWith(secondTree->rightLeaf);
            if (rightLeaf != NULL)
                result &= rightLeaf->isMirrorResemlanceWith(secondTree->leftLeaf);
        }
                }
                return result;
  }

bool isEqualWith(std::unique_ptr<BinaryTree>& secondTree)
{
    bool result = false;
    if (secondTree != NULL)
    {
        if ( (secondTree->leftLeaf  == NULL) == (leftLeaf  == NULL) &&
             (secondTree->rightLeaf == NULL) == (rightLeaf == NULL) &&
             (secondTree->value == value)                           ){
            result = true;
            if (leftLeaf != NULL)
                result = leftLeaf->isEqualWith(secondTree->leftLeaf);
            if (rightLeaf != NULL)
                result &= rightLeaf->isEqualWith(secondTree->rightLeaf);
        }
                }
```

```
            return result;
    }

    bool isSymmetryWith(std::unique_ptr<BinaryTree>& secondTree)
    {
        bool result = false;
        if (secondTree != NULL)
        {
            if ( (secondTree->leftLeaf  == NULL) == (rightLeaf  == NULL) &&
                 (secondTree->rightLeaf == NULL) == (leftLeaf   == NULL) &&
                 (secondTree->value == value)                           ){
                result = true;
                if (leftLeaf != NULL)
                    result = leftLeaf->isSymmetryWith(secondTree->rightLeaf);
                if (rightLeaf != NULL)
                    result &= rightLeaf->isSymmetryWith(secondTree->leftLeaf);
            }
                    }
                    return result;
    }

    ~BinaryTree() = default;

private:

    int getEndPos(std::string str, int startPos)
    {
        int endPos;
        if (str[startPos + 1] == '(')
        {
            endPos = startPos + 2;
            int brackets = 1;
            while ((brackets > 0) && (endPos < str.length()))
            {
                if (str[endPos] == '(') brackets++;
                if (str[endPos] == ')') brackets--;
                endPos++;
            }
            if (brackets > 0) {
                            std::cout<<"Неправильная запись дерева" << std::endl;
                            exit(1);
                    } // неправильная строка
            endPos--;
        }
        else
            endPos = startPos;
        return endPos;
    }
};
```